Lab 10 tasks

In this lab you are going to implement multi-thread client-server UDP application for remote shell.

Task 0

Run the Client-Server python application that we studied in the lecture, and send “Hi” message to Server. Create some groups, and send broadcast message to the group members.

Task 1 – private remote shell

Client should run mount command to mount Server file system be visible to Client.
After mounting, Client may run any commands on Server file system.
For example, after the client cd /Server (remote file system of Server), the client can ls, and get the content of the /Server folder. Indeed, the client can run any shell command on Server’s file system, and should get the result printed locally on Client’s screen.

```
/local/usr $ mount private host:port:/Server
/local/usr $   cd host:port:/Server
/Server $   cd usr
/Server/usr $   ls
     a.txt    b.txt
/Server/usr $
```

Note: till the client does not cd /Server file system, all the commands are executed on Clients (local) file system. If the client changes directory (cd) back to some local directory, all the shell commands should be run locally.

You may find the following links useful:

https://www.geeksforgeeks.org/os-walk-python/

https://stackoverflow.com/questions/4760215/running-shell-command-and-capturing-the-output

Task 2 – get file from Server file system

Add functionality “get <file name> <local path>” to the server so that a client can ask to transfer some file from Server file system. The client would get the copy of this file in its local path of clients file system.

Run example:
(cwd in “cp a.txt cwd” means current working directory on client’s machine)

```
/local/usr $ mount private host:port:/Server
/local/usr $   cd host:port:/Server
/Server $   cd usr
/Server/usr $   ls
     a.txt    b.txt
/Server/usr $   cp a.txt cwd
/Server/usr $   cd local:/local/usr
/local/usr $ ls
    a.txt
/Server/usr $
```

Task 3 – shared remote shell

Add functionality to the server so that multiple clients can share remote shell.

When some command is sent to the server, the server sends this command to all active (i.e., logged in) clients that did not send this command. Server executes the command on its file system, and then all clients get the output of this command.

Running example:

Client A:

```
/local/usr $ mount shared host:port:/Server
/local/usr $   cd host:port:/Server
/Server $   cd usr
/Server/usr $   ls
     a.txt    b.txt
/Server/usr $  (client B):cat a.txt
Hello world !
/Server/usr $
```

Client B:

```
/local/usr $ mount shared host:port:/Server
/local/usr $   cd host:port:/Server
/Server $   (client A):cd usr
/Server/usr $  (client A):ls
     a.txt    b.txt
/Server/usr $   cat a.txt
Hello world !
/Server/usr $
```

Note: assume correct multi-process interleaving, i.e., assume that two processes do not run commands at the same time, but sequentially.

Task 4 – bonus – sharing movies files between clients

Add functionality to the server so that the clients can share their files.

Each client , when it is connected to the server, may run “share <directory name>”.

This means that this client would like to share all the files from this directory with other clients.

When some client asks the server “find <file name>”, the server should look for this file at all clients that have shared directory. For this, server sends request package to all the clients that have shared directory, to execute search on their shared directory. The clients should run the search and return to server “true” or “false” response. Server would send to one of the clients with the requested file to send this file to the client that asked for it.

We assume that two files with the same name on two clients’ file systems are indeed the same file (for example, files are movies, so the file with the same name is the same movie).

Good luck !